Developing UPPAAL over 15 Years

Gerd Behrmann[1], Alexandre David[2], Kim Guldstrand Larsen[2], Paul Pettersson[3], and Wang Yi[4]

[1] NORDUnet A/S, Copenhagen, Denmark
[2] Department of Computer Science, Aalborg University, Denmark
[3] Mälardalen Research and Technology Centre, Mälardalen University, Sweden
[4] Department of Information Technology, Uppsala University, Sweden
behrmann@ndgf.org, {adavid,kgl}@cs.aau.dk, paul.pettersson@mdh.se, yi@it.uu.se

Abstract. UPPAAL is a tool suitable for model checking real-time systems described as networks of timed automata communicating by channel synchronizations and extended with integer variables. Its first version was released in 1995 and its development is still very active. It now features an advanced modelling language, a user-friendly graphical interface, and a performant model checker engine. In addition, several flavors of the tool have matured in recent years. In this paper, we present how we managed to maintain the tool during 15 years, its current architecture with its challenges, and we give future directions of the tool.

1 Development History

UPPAAL is first of all a research tool born from the collaboration of Uppsala and Aalborg universities [24]. Its theory comes from [1] with decidability results based on regions. Its performance originally comes from zones [18] as a representation for states. Since then the development has been fuelled by scientific results on algorithms or new data structures such as [4,5,6,9,19,20] and very importantly by case studies that pushed us to push the limits of the tool, such as [7,10,11,21,22]. On the other hand, having such a tool helps to develop and test new theories and algorithms, which has given us a synergy during the last decade between tool development and publications.

Recently, the tool has blossomed into several domain specific versions, namely, CORA [5] (cost-optimal reachability), TRON [17] (online testing), COVER [15,16] (coverage testing), TIGA [3] (timed game solver), PORT [13,14] (component based), PRO (extension with probabilities, in progress), and TIMES [12,2] (scheduling and analysis). These extensions are made based on a common code base, re-using basic data structures to represent states, store them, and perform some basic computations.

How have we managed to get going for 15 year across different physical sites with changing teams? The first reason is our commitment to have an efficient tool implementing our research results. A tool strengthens and sometimes disprove theories. Second, we use a centralized version management system (cvs and then

subversion), which allows distributed teams to work on the same code. A given checkout of the repository contains all variants of the tool but they all live in their own separated modules. Developers are responsible for few modules and modify other modules occasionally only. Finally, we are using a bug management system (bugzilla) and we do regression testing. We update our battery of tests with examples that trigger new bugs. To find which changes in the repository history trigger a new bug, we use binary search on the revision numbers until we find a revision n where the bug is not present and a revision $n + 1$ where it is present. This is a very effective technique.

In the long term, the code base goes through different life cycles. The first cycle was with the original atg graph editor and an early custom simulator. The second introduced an integrated graphical editor, the client-server architecture still in use today, and an improved engine. The third cycle is the current one with a modular pipeline architecture. The development is incremental during a cycle, following the current design and making changes until the amount of desired features and new algorithms reaches a threshold. Then there is a major effort to re-design or re-factor the code and we continue. The current architecture has lived up to its expectations for approximately 8 years, during which we could re-use existing components and create new ones that we could litteraly plug together. However, now is the time for a major update.

2 Current Architecture Overview

Uppaal is based on a client-server architecture with the graphical interface (client) communicating with the model checker (server) via a local pipe or the network. This separation of concerns makes Uppaal easier to port and maintain on different platforms.

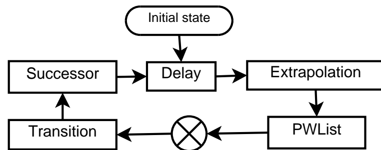

Fig. 1. Simplified pipeline architecture.

The model checker itself is designed around a pipeline architecture [4] where each block or *filter* processes states and sends them to the next stage as shown in Fig. 1. The different stages include, e.g., delay, extrapolation, or storing states. Typically the reachability analysis pipeline has a while loop taking states from our (unified) passed and waiting list structure and explores them by pushing them to the first filter. The chain is *Transition* (which transitions can be taken) - *Successor* (execution of the transitions) - *Delay* (let time pass) - *Extrapolation* (apply an appropriate extrapolation to ensure finite exploration) - *PWList* (inclusion check and mark the state to be explored) - *Query* (evaluate the formula if the state was not included). Implementing another checker, e.g. a timed game solver, is relatively easy and consists in adding components that will do the backward propagation, changing the first filter to either explore forward or backward, add a post-processing filter to detect what is winning or losing in the game after *Extrapolation*, and changing the graph representation. To change the semantics of the game, e.g. to implement

simulation checking [8], mainly consists in changing *Transition* that implements the transition relation and changing *Delay* to allow turn-based delay.

In addition to these components, Uppaal contains a virtual machine to execute the compiled byte-code of our C-like input language supporting user defined functions and types. This is abstracted in the form of *Expression* objects that we can re-use across different flavours of Uppaal, which makes other extensions such as adding probabilities easier.

We currently distribute some open source components, such as the parser and the difference bound matrix (DBM) library. The parser understands the XML format we use in Uppaal, which allows other researchers to use the same format. The DBM library handles DBMs and federations (unions of DBMs) used to represent symbolic states. The DBM library supports a wide range of operations including subtractions and merging of DBMs.

3 Challenges

The current architecture has been pushed to implement the different known flavours of Uppaal but also to extend every checker. Recent extensions to Uppaal include priorities and stop-watches. Tiga was recently extended with a simulation checker. It is being extended with a new timed interface checker. Although the overall pipeline architecture accommodates these extensions, we have reached the limit of some "implementation details". These are: 1) there can be only one global system, 2) long wished features, such as clock constraints on receiving edges of broadcast synchronizations, are now needed, 3) the engine is designed for 32-bit architectures, 4) there is no multi-core support, 5) there is only one kind of symbolic state, and the list goes on.

Updating to 64-bit is mainly technical. Going for multi-core support (multi-threaded Uppaal) is more challenging. There have been experiments in the past in this direction and we know that the current architecture could be adapted by having one thread per pipeline copy. This fits memory locality but we also know that it did not work so well because blocking data-structures (access protected by mutex) were major bottlenecks. It is crucial to have non-blocking structures such as [23]. In addition, we want to make the components extendable more easily in particular to allow more people to work on Uppaal without having to know what most of the code is doing.

4 Future

Uppaal has already spawned one company, UP4ALL[5], that sells a version of the tool for commercial uses. Another market we intend to target is testing. Research tools really have a future if they can be applied and used outside academia, as witnessed by Lustre/SCADE. The current trend of our research is to explore different domains as the different flavors of Uppaal show. That also means

[5] To contact UP4ALL email *sales@uppaal.com*.

that a new life-cycle with another architectural revision is now needed to cope with more extensions of UPPAAL. That will enable us to let other researchers experiment with the internals of UPPAAL and still maintain our core engine.

5 Acknowledgements

It is important to remember that UPPAAL is the result of the cumulative efforts of many collaborators. Among them we would like to thank early pioneers Johan Bengtsson and Fredrik Larsen, former contributor of the graphical interface Carsten Weise, and active contributor and maintainer Marius Mikučionis (UPPAAL and TRON). We also thank contributors of different extensions of UPPAAL, among them Didier Lime (TIGA), John Håkansson (PORT), Anders Hessel (COVER), Leonid Mokrushin (TIMES), Jakob Illum (CORA), Arild Haugstad (PRO).

References

1. R. Alur and D. L. Dill. Automata for Modeling Real-Time Systems. In *Proc. of ICALP*, volume 443 of *LNCS*, pages 322–335, 1990.
2. T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for modelling and implementation of embedded systems. In J.-P. Katoen and P. Stevens, editors, *Proc. of the 8*th *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 2280 in Lecture Notes in Computer Science, pages 460–464. Springer–Verlag, 2002.
3. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *CAV'07*, number 4590 in LNCS, pages 121–125. Springer, 2007.
4. G. Behrmann, A. David, K. G. Larsen, and W. Yi. Unification & Sharing in Timed Automata Verification. In *SPIN Workshop 03*, volume 2648 of *LNCS*, pages 225–229, 2003.
5. G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, and J. Romijn. Efficient Guiding Towards Cost-Optimality in UPPAAL. In T. Margaria and W. Yi, editors, *Proc. of the 7th Int. Conf. on TACAS*, number 2031 in LNCS, pages 174–188. Springer-Verlag, 2001.
6. G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *Proc. of the 12th Int. Conf. on CAV*, volume 1633 of *LNCS*. Springer, 1999.
7. J. Bengtsson, W. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an Audio Protocol with Bus Collision Using UPPAAL. In R. Alur and T. A. Henzinger, editors, *CAV96*, number 1102 in LNCS, pages 244–256. Springer-Verlag, July 1996.
8. P. Bulychev, T. Chatain, A. David, and K. G. Larsen. Efficient on-the-fly Algorithm for Checking Alternating Timed Simulation. In *FORMATS'09*, number 5813 in LNCS, pages 73–87. Springer, 2009.
9. A. David, J. Håkansson, K. G. Larsen, and P. Pettersson. Model Checking Timed Automata with Priorities using DBM Subtraction. In *Proc. of the 4th Int. Conf. on FORMATS*, volume 4202 of *LNCS*, pages 128–142, 2006.

10. A. David, M. O. Möller, and W. Yi. Formal Verification of UML Statecharts with Real-Time Extensions. In R.-D. Kutsche and H. Weber, editors, *FASE, 5th Int. Conf. 2002*, volume 2306 of *LNCS*, pages 218–232. Springer, 2002.
11. A. David and W. Yi. Modelling and Analysis of a Commercial Field Bus Protocol. In *Proc. of Euromicro-RTS'00*, pages 165–172. IEEE Computer Society, 2000.
12. E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In J.-P. Katoen and P. Stevens, editors, *Proc. of the 8*th *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 2280 in Lecture Notes in Computer Science, pages 67–82. Springer–Verlag, 2002.
13. J. Håkansson, J. Carlson, A. Monot, P. Pettersson, and D. Slutej. Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT. In S. D. Cha, J.-Y. Choi, M. Kim, I. Lee, and M. Viswanathan, editors, *ATVA*, volume 5311 of *LNCS*, pages 252–257. Springer, 2008.
14. J. Håkansson and P. Pettersson. Partial Order Reduction for Verification of Real-Time Components. In *Proc. of the 5th Int. Conf. on FORMATS*, LNCS. Springer-Verlag, 2007.
15. A. Hessel and P. Pettersson. A Test Case Generation Algorithm for Real-Time Systems. In H.-D. Ehrich and K.-D. Schewe, editors, *Proc. of the Fourth ICQS*, pages 268–273. IEEE Computer Society, 2004.
16. A. Hessel and P. Pettersson. Cover — A Test-Case Generation Tool for Timed Systems. In A. Petrenko, M. Veanes, J. Tretmans, and W. Grieskamp, editors, *Testing of Software and Communicating Systems: Work-in-Progress and Position Papers, Tool Demonstrations, and Tutorial Abstracts of TestCom/FATES*, pages 31–34, 2007.
17. K. Larsen, M. Mikučionis, and B. Nielsen. Online Testing of Real-time Systems Using Uppaal. In *FATES'04*, LNCS, Linz, Austria, September 2004.
18. K. G. Larsen, P. Pettersson, and W. Yi. Model-Checking for Real-Time Systems. In *Proc. of Fundamentals of Computation Theory*, number 965 in LNCS, pages 62–88, August 1995.
19. F. Larsson, K. G. Larsen, P. Pettersson, and W. Yi. Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction. In *Proc. of the 18th IEEE RTSS*, pages 14–24. IEEE Computer Society Press, December 1997.
20. F. Larsson, P. Pettersson, and W. Yi. On Memory-Block Traversal Problems in Model Checking Timed Systems. In S. Graf and M. Schwartzbach, editors, *Proc. of the 6th Conf. on TACAS*, number 1785 in LNCS, pages 127–141. Springer-Verlag, 2000.
21. M. Lindahl, P. Pettersson, and W. Yi. Formal Design and Analysis of a Gear-Box Controller. In *Proc. of the 4th Workshop on TACAS*, number 1384 in LNCS, pages 281–297. Springer-Verlag, March 1998.
22. H. Lönn and P. Pettersson. Formal Verification of a TDMA Protocol Startup Mechanism. In *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems*, pages 235–242, December 1997.
23. C.-H. Shann, T.-L. Huang, and C. Chen. A practical nonblocking queue algorithm using compare-and-swap. In *Seventh International Conference on Parallel and Distributed Systems*, pages 470–475, 2000.
24. W. Yi, P. Pettersson, and M. Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In D. Hogrefe and S. Leue, editors, *Proc. of FORTE'94*, pages 223–238. North–Holland, 1994.